Les web services avec JAX-WS 2.0

JAX-WS est la nouvelle appellation de JAX-RPC (Java API for XML Based RPC) qui permet de développer très simplement des services web. JAX-WS fournit un ensemble d'annotations pour mapper la correspondance Java-WSDL. Il suffit pour cela d'annoter directement les classes Java qui vont représenter le service web. En ce qui concerne le client, JAX-WS permet d'utiliser une classe proxy pour appeler un service distant et masquer la complexité du protocole. Ainsi, ni le client ni le serveur n'ont besoin de générer ou de parser les messages SOAP. JAX-WS s'occupe de ces traitements de bas niveau.

Dans l'exemple ci-dessous, une classe Java utilise des annotations JAX-WS qui vont permettre par la suite de générer le document WSDL. Le document WSDL est auto-générer par le serveur d'application au moment du déploiement :

```
@WebService()
public class Banque {
@WebMethod
public double conversionEuroToDh(double mt) {
...
}
...
}
```

JAX-WS s'appuie sur l'API JAXB 2.0 pour tout ce qui concerne la correspondance entre document XML et objets Java. Un graphe d'objets est constitué suivant les éléments constituant le document XML. JAXB (Java Architecture for XML Binding) facilite la correspondance bidirectionnelle en fournissant un niveau d'abstraction plus élevé que SAX ou DOM et en s'appuyant sur les annotations.
Par exemple, si nous désirons obtenir une représentation XML de la classe Client, il suffit de l'annoter avec @javax.xml.bind.annotation.XmlRootElement. D'autres annotations permettent de spécifier qu'un attribut est un identifiant avec **@XmlID** ou de renommer un attribut (e-mail au lieu de email, par exemple) avec **@XmlAttribute** :

```
@XmlRootElement
public class Client {
@XmlID
private String id;
private String nom;
private String prénom;
@XmlAttribute(name="e-mail")
private String email;
}
```

Ces annotations permettent alors de générer le document XML suivant à partir de la classe, et inversement :

```
<?xml version="1.0" encoding="UTF-8"?>
<client>
     <id>3456</id>
     <nom>REMY</nom>
     <prénom>Emmanuel</prénom>
     <e-mail>emmanuel.remy@orange.fr</e-mail>
</client>
```

Mapping entre WSDL et JAX-WS/JAXB :

Le document WSDL qui est généré par le serveur d'applications respecte le standard XML Schema. Ce standard permet de définir précisément les types de chaque élément constituant le document XML. Ces types sont important puisqu'ils définissent les paramètres de chaque méthode de la classe qui constitue le service web. Ainsi le mapping entre une classe Java est son correspond XML peut se faire correctement grâce à JAXB en respectant les types proposés.
Vous avez justement ci-dessous la correspondance entre les types XML Schema et les types Java:

XML Schema	**Types Java**
xsd:byte	Byte
xsd:Boolean	Boolean
xsd:short	Short
xsd:int	Integer
xsd:long	Long
xsd:float	Float
xsd:double	Double
xsd:string	java.lang.String
xsd:dateTime	java.util.Calendar
xsd:integer	java.math.BigInteger
xsd:decimal	java.math.BigDecimal
xsd:QName	java.xml.namespace.QName
xsd:base64Binary	Byte [ ]
xsd:hexBinary	Byte [ ]

Vous remarquez que le mapping proposé par le couple JAX-WS/JAXB utilise systématiquement les classes enveloppes plutôt que les types primitifs. Vous pouvez malgré tout utiliser les types primitifs pour les paramètres de vos méthodes puisque l'auto-boxing entre la classe enveloppe et son type primitif se réalise automatiquement, sauf pour les tableaux d'octets. En effet, un Byte[] est totalement différent d'un byte[].
Généralement, la définition de tous les types utilisés par le service web, notamment par les paramètres des méthodes accessibles sont placés dans un fichier à part et qui correspond justement au Schema prévu. Ainsi, nous aurons donc deux fichiers, le fichier WSDL qui porte l'extension <*.wsdl> qui fera lui même référence au fichier XML Schema dont l'extension est généralement <*.xsd>.

Comment développer un service web avec JAX-WS.

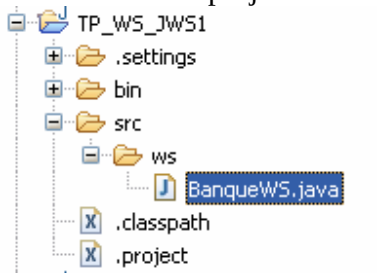
En réalité, le développement d'un service web est relativement simple bien que plusieurs technologies soient mises en oeuvre. Telle est la force de Java EE. Pour cela, il existe plusieurs phases de génération de code qui entre en jeu et qui mettent en oeuvre toute la tuyauterie technique. Le développement et l'utilisation d'un service web comporte quatre phases :

1. Développement du service web proprement dit.
2. Génération des artefacts côté serveur (Skeletons).
3. Génération des artefacts côté client (Stubs).
4. Démarrer le serveur en déployant le web service.
5. Appel du service web chez le client.

Un artefact est composé de l'ensemble des documents nécessaires à un service web. Nous pouvons citer par exemple le document WSDL ou encore les classes Java qui formeront les messages SOAP d'échanges XML.

1- Développer un service web :

- Outils à installer :
  - JDK1.6
  - Editeur java : Eclipse.
- Créer un projet java en utilisant JDK1.6 comme compilateur et environnement d'exécution java.
- Structure du projet :



- Créer la classe BanqueWS.java du package ws :

```
package ws;

import java.util.Date;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService(targetNamespace="http://bk/test",
            serviceName="BanqueService")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
            use=SOAPBinding.Use.LITERAL)
public class BanqueWS {
      private double tauxDeChange=11;
```

```
	@WebMethod(operationName="conversion")
	public double conversionDhEuro(@WebParam(name="montant")double
mt){
		return tauxDeChange*mt;
	}
	@WebMethod(operationName="changeTaux")
	public boolean changeTaux(double taux){
		this.tauxDeChange=taux;
		return true;
	}
	@WebMethod(operationName="dateDuServeur")
	public Date getServerDate(){
		return(new Date());
	}

}
```

- Cette classe défini un web service nommé BanqueService qui est défini par une variable d'instance `tauxDeChange` et trois méthodes :
  - Méthode nommée conversion qui permet de retourner un montant converti de l'euro en dh en utilisant le taux de change courant.
  - Une méthode qui permet de modifier le taux de change
  - Une méthode qui permet de retourner la date du serveur.

Comme nous l'avons découvert dans le chapitre précédent, les annotations JAX-WS sont spécifiques aux services web. Elles permettent d'agir sur la structure d'un document WSDL en modifiant certains paramètres du service ou des méthodes qui le compose. Dans cette rubrique, nous allons décrire plus précisément le comportement de ces annotations.

- L'annotation principale pour définir un service web est @javax.jws.WebService. Cette annotation s'applique à une classe et utilise éventuellement plusieurs attributs qui permettent, par exemple, de spécifier la localisation d'un service web.

Voici La liste des attributs qu'il est possible de mettre en oeuvre :

1- **name** : définit le nom du service web. Ce nom se trouve alors dans le fichier WSDL dans l'attribut name de la balise <portType>. La valeur par défaut est le nom de la classe d'imlémentation, ici BanqueWS.

2- **serviceName** : définit le nom du service. Ce nom se trouve dans le fichier WSDL dans l'attribut name de la balise <service>. La valeur par défaut est le nom de la classe d'implémentation suivie du suffixe "Service", ici donc BanqueWSService.
3- **wsdlLocation** : définit l'URL, qu'elle soit relative ou absolue, d'un fichier WSDL existant. Par défaut, ce fichier est auto-généré lors du déploiement.
4- **endpointInterface** : définit le nom complet de l'interface "endpoint" définissant les méthodes du service web. Cela permet au développeur de séparer le "contrat" (l'interface) de l'implémentation. L'implémentation de cette interface par le service n'est pas requise. L'intérêt de cette solution est également d'affecter des annotations de mapping Java vers le WSDL dans l'interface et non dans la classe d'implémentation.

- L'annotation **@WebMethod** est utilisée pour déclarer les méthodes accessibles par le service web (nous parlons d'opération dans le protocole SOAP). Voici les détails des attributs qu'elle contient :
  - **operationName** : définit la valeur de l'attribut name de la balise <operation> dans le fichier WSDL. Celui-ci représente le nom de l'opération.
  - **exclude** : marque une méthode comme étant non disponible par le service web. Ce paramètre est utilisé lorsque nous souhaitons masquer une méthode héritée, par exemple.
- Une méthode qui n'a pas de paramètre de retour peut être annotée par **@Oneway**.

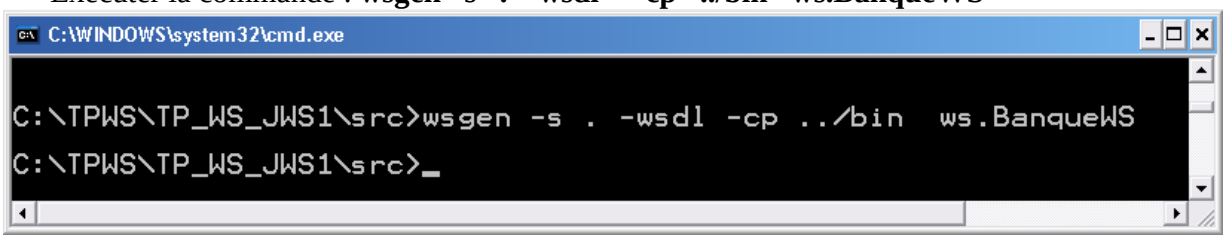
- Les paramètres de la méthode ainsi que la valeur de retour peuvent aussi être changées. L'annotation **@javax.jws.WebParam** permet de contrôler la génération du WSDL qui concerne les paramètres de la méthode. Ainsi, l'annotation **@WebParam** permet de préciser des informations concernant les arguments des opérations définies dans le fichier WSDL. Voici le détail des attributs disponibles :

  - **name** : définit le nom du paramètre qui sera utilisé dans le fichier WSDL. Par défaut, le nom de l'argument (du code Java) est utilisé.
  - **header** : indique si la définition du paramètre se trouve dans l'en-tête (header) plutôt que dans le corps (body). La valeur par défaut étant false, la définition se trouve donc dans le corps.
  - **mode** : indique si le paramètre est utilisé en entrée, en sortie ou les deux. Pour spécifier ce type, il faut utiliser l'énumération WebParam.MODE (Valeurs : IN, OUT, BOTH).
  - **targetNamespace** : définit le namespace à utiliser. Par défaut, nous utilisons un namespace vide. Cet attribut est à utiliser si le paramètre est mappé dans l'en-tête.
- L'annotation **@WebResult** est presque identique, mais elle annote la valeur de retour de la méthode.

2- Génération des artefacts côté serveur (Skeletons).

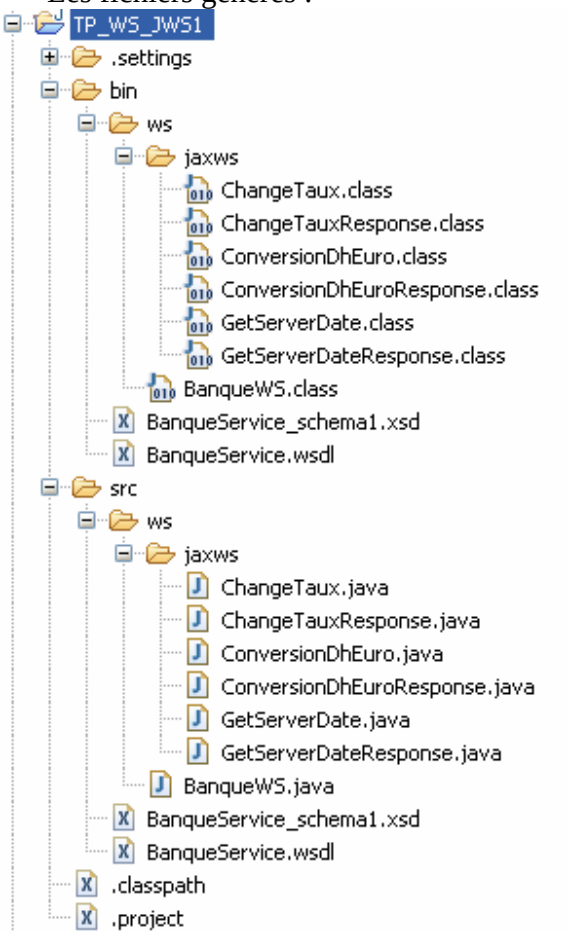
Pour générer les artefacts du web service coté serveur, vous pouvez utiliser l'utilitaire wsgen fourni par jdk1.6. Nous allons faire cette opération sur ligne de commande :

- Vérifier si la variable d'environnement path pointe c:\jdk1.6\bin
- Sur ligne de commande, il faut se placer dans le dossier src de votre projet
- Exécuter la commande : **wsgen –s . –wsdl –cp ../bin ws.BanqueWS**



- L'option –s suivie de **.** , indique à wsgen de placer les fichier source (.java) dans le dossier courant : src.
- L'option –wsdl peut être utilisé pour générer le wsdl du web service. Ce qui n'est pas obligatoire, car le serveur le génère automatiquement à la demande du client.
- L'option –cp suivie de ../bin indique à wsgen que la classe ws.BanqueWS se trouve dans le dossier bin du projet.

- Les fichiers générés :



- Pour chaque méthode du web service, l'utilitaire wsgen, génère deux classes java qui permettent de sérialiser et de désérialiser les messages SOAP.
- Le wsdl généré est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://bk/test" name="BanqueService"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://bk/test"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://bk/test"
schemaLocation="BanqueService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="conversion">
    <part name="parameters" element="tns:conversion"/>
  </message>
  <message name="conversionResponse">
    <part name="parameters" element="tns:conversionResponse"/>
  </message>
  <message name="changeTaux">
    <part name="parameters" element="tns:changeTaux"/>
  </message>
```

```
  <message name="changeTauxResponse">
    <part name="parameters" element="tns:changeTauxResponse"/>
  </message>
  <message name="dateDuServeur">
    <part name="parameters" element="tns:dateDuServeur"/>
  </message>
  <message name="dateDuServeurResponse">
    <part name="parameters" element="tns:dateDuServeurResponse"/>
  </message>
  <portType name="BanqueWS">
    <operation name="conversion">
      <input message="tns:conversion"/>
      <output message="tns:conversionResponse"/>
    </operation>
    <operation name="changeTaux">
      <input message="tns:changeTaux"/>
      <output message="tns:changeTauxResponse"/>
    </operation>
    <operation name="dateDuServeur">
      <input message="tns:dateDuServeur"/>
      <output message="tns:dateDuServeurResponse"/>
    </operation>
  </portType>
  <binding name="BanqueWSPortBinding" type="tns:BanqueWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="conversion">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="changeTaux">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="dateDuServeur">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="BanqueService">
    <port name="BanqueWSPort" binding="tns:BanqueWSPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
```

```
    </port>
  </service>
</definitions>
```

- Ce fichier fait ensuite référence à la définition des types qui sont décrit dans le fichier correspondant au XML Schema et qui porte l'extension <.xsd>. Ce fichier est également auto-généré par l'utilitaire wsgen. Le contenu de ce fichier nommé **BanqueService_schema1.xsd** est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://bk/test"
xmlns:tns="http://bk/test" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="changeTaux" type="tns:changeTaux"/>
  <xs:element name="changeTauxResponse" type="tns:changeTauxResponse"/>
  <xs:element name="conversion" type="tns:conversion"/>
  <xs:element name="conversionResponse" type="tns:conversionResponse"/>
  <xs:element name="dateDuServeur" type="tns:dateDuServeur"/>
  <xs:element name="dateDuServeurResponse"
type="tns:dateDuServeurResponse"/>
  <xs:complexType name="dateDuServeur"/>
  <xs:complexType name="dateDuServeurResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:dateTime" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="conversion">
    <xs:sequence>
      <xs:element name="montant" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="conversionResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="changeTaux">
    <xs:sequence>
      <xs:element name="arg0" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="changeTauxResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

3- Création du serveur.

- Notre web service est prêt à être déployé dans un serveur d'application J2EE. Dans ce TP, nous allons créer notre propre serveur.
- Dans le dossier src, créer une classe ServeurJWS.java dont le code est le suivant :

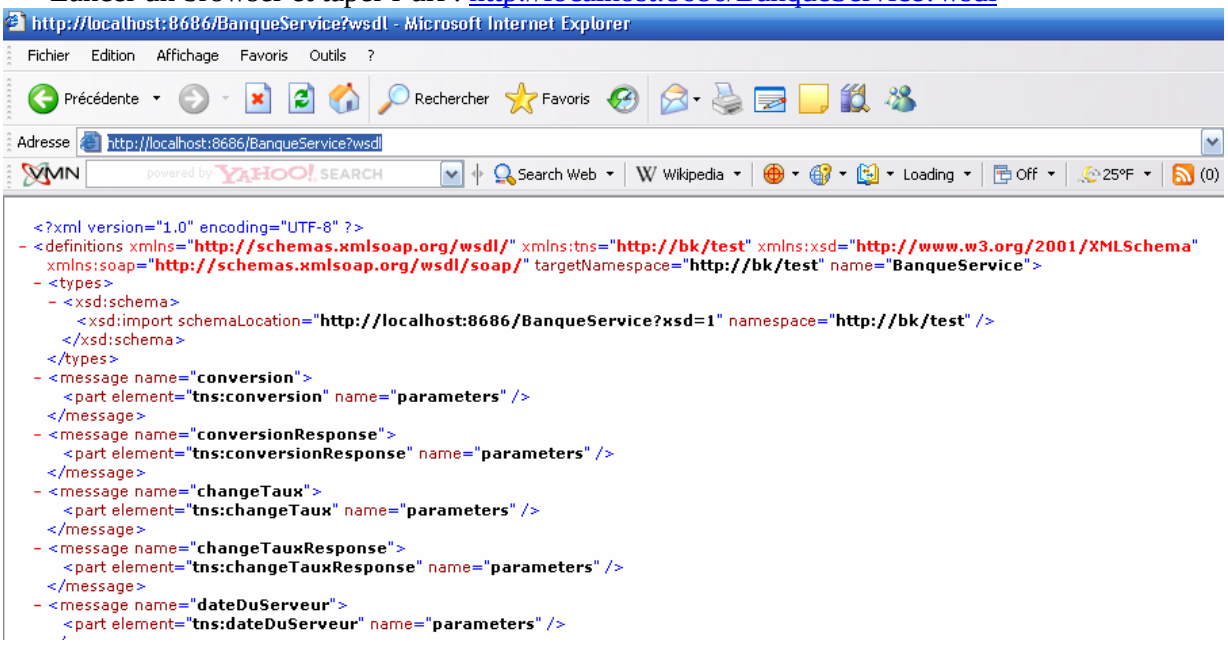
```
import javax.xml.ws.Endpoint;
import ws.BanqueWS;
public class ServeurJWS {

	public static void main(String[] args) {
		String uri="http://localhost:8686/";
		Endpoint.publish(uri, new BanqueWS());
		System.out.println("Web Service publié à "+uri);
	}

}
```

- Démarrer le serveur en exécutant l'application ServeurJWS. Le serveur démarre en affichant le message suivant :

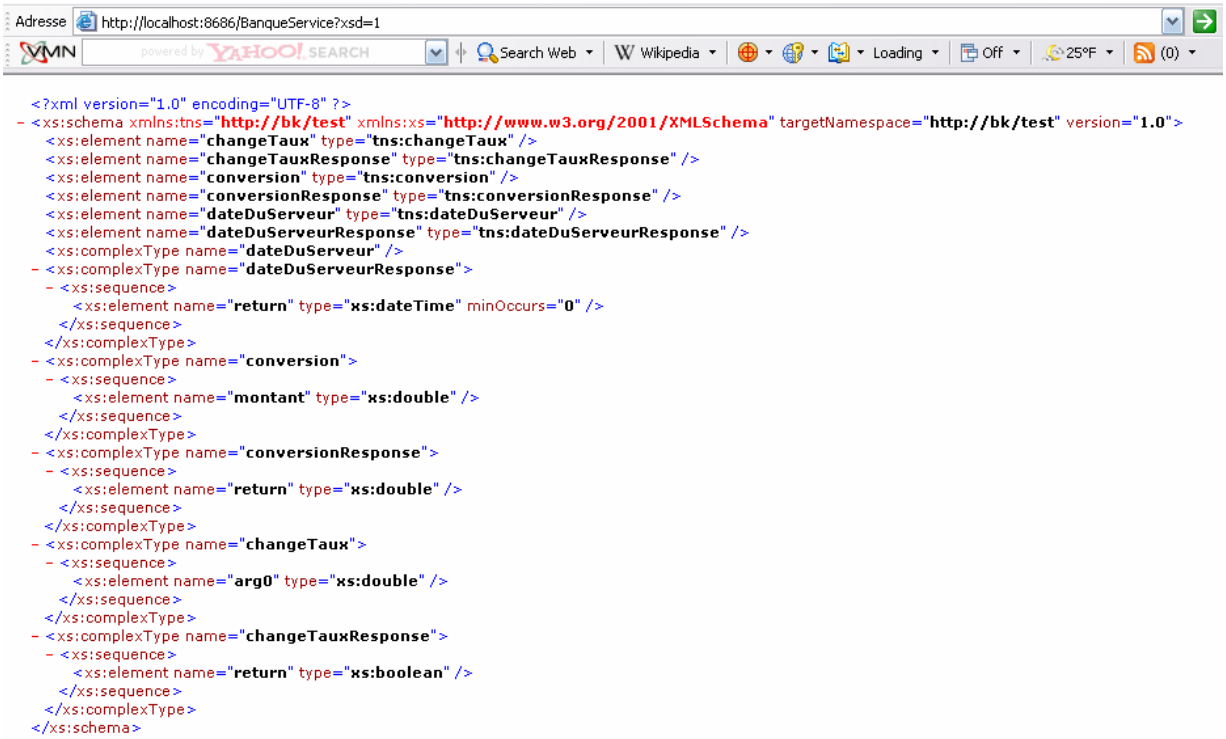
```
Web Service publié à http://localhost:8686/
```

4- Tester le web service avec un navigateur web

- Lancer un browser et taper l'url : http://localhost:8686/BanqueService?wsdl

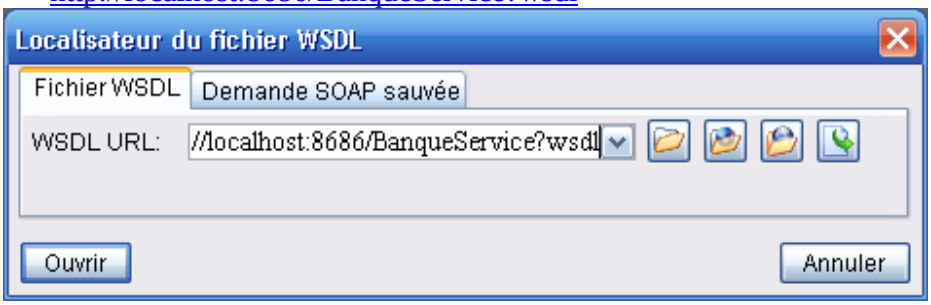


- Pour visualiser le schéma XML des messages SOAP, taper l'url : http://localhost:8686/BanqueService?xsd=1
-

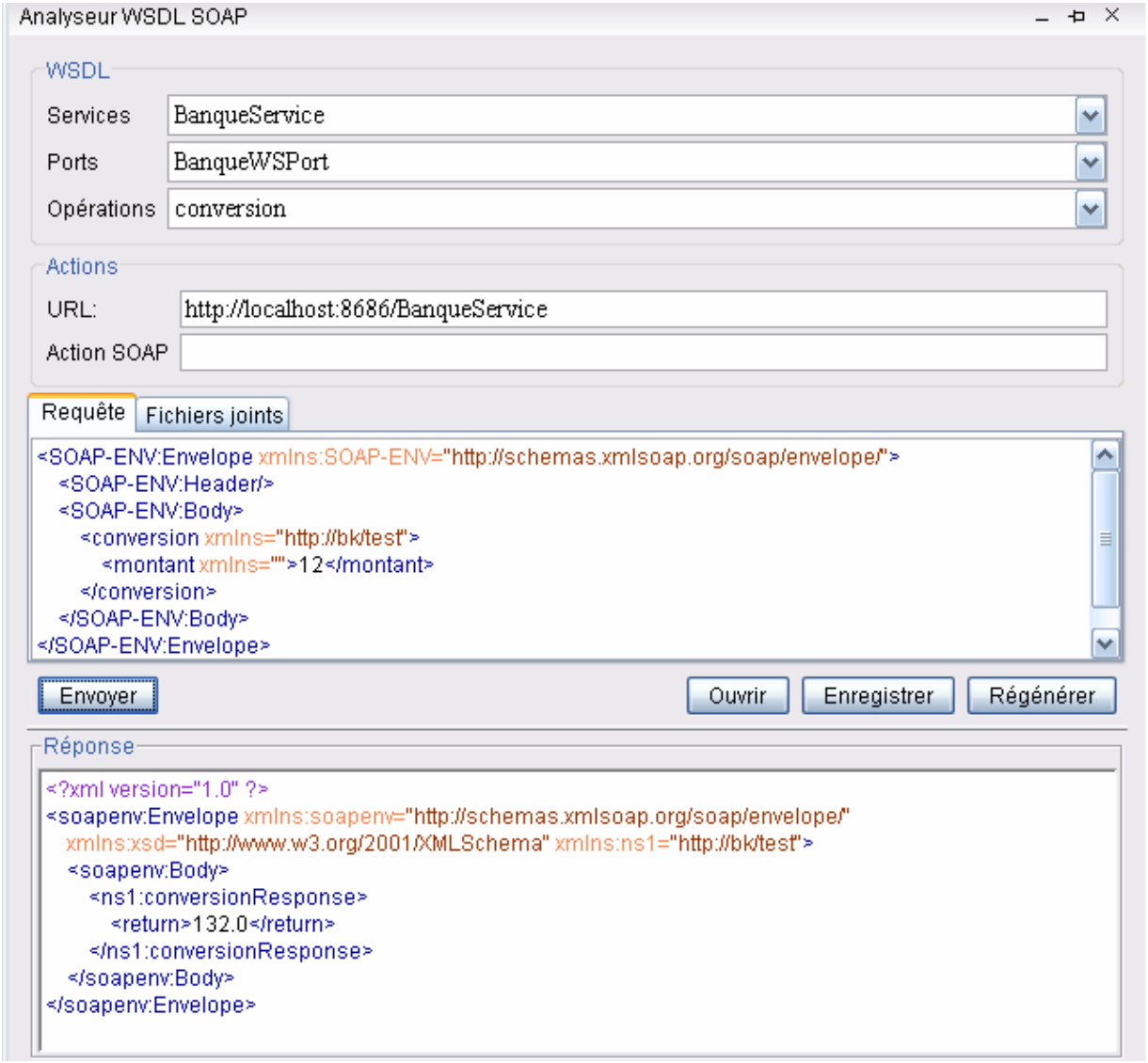



```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:tns="http://bk/test" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://bk/test" version="1.0">
    <xs:element name="changeTaux" type="tns:changeTaux" />
    <xs:element name="changeTauxResponse" type="tns:changeTauxResponse" />
    <xs:element name="conversion" type="tns:conversion" />
    <xs:element name="conversionResponse" type="tns:conversionResponse" />
    <xs:element name="dateDuServeur" type="tns:dateDuServeur" />
    <xs:element name="dateDuServeurResponse" type="tns:dateDuServeurResponse" />
    <xs:complexType name="dateDuServeur" />
  - <xs:complexType name="dateDuServeurResponse">
    - <xs:sequence>
        <xs:element name="return" type="xs:dateTime" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="conversion">
    - <xs:sequence>
        <xs:element name="montant" type="xs:double" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="conversionResponse">
    - <xs:sequence>
        <xs:element name="return" type="xs:double" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="changeTaux">
    - <xs:sequence>
        <xs:element name="arg0" type="xs:double" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="changeTauxResponse">
    - <xs:sequence>
        <xs:element name="return" type="xs:boolean" />
      </xs:sequence>
    </xs:complexType>
</xs:schema>
```

- Pour tester les méthodes, nous avons besoin d'un analyseur WSDL SOAP. La majorité des éditeurs XML offre ce genre d'analyseur
- Lancer l'éditeur xml Oxygen
- Dans le menu « outils », lancer l'analyseur WSDL SOAP.
- Dans la zone de texte WSDL URL de la fenêtre qui apparaît, taper l'adresse du wsdl http://localhost:8686/BanqueService?wsdl



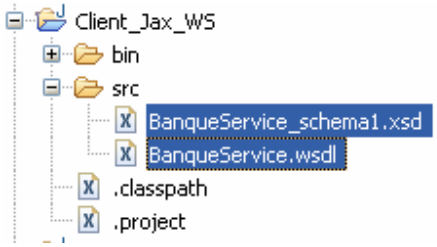
- Cliquez sur le bouton Ouvrir



- Cet outil est très pratique car il vous permet de sélectionner l'opération que vous voulez appeler via SOAP et taper dans la requête SOAP, les paramètre de la méthode. Et en cliquant sur le bouton Envoyer, la requête SOAP est envoyée au web service et la réponse SOAP s'affiche en bas de la requête SOAP
- Essayer de tester toutes les opérations du web service.

5- Création d'un Client Java.

- Créer un nouveau projet java projet java pour le client.
- Dans le dossier src, placer les deux fichier wsdl et XSD du web service.

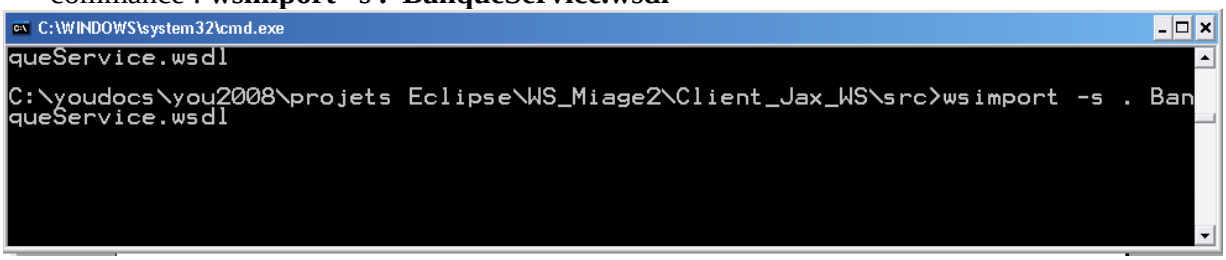


- Avant de créer l'application du client, nous avons besoin de générer les interfacts coté client (STUBS) à partir du wsdl. Cette opération peut être effectuée en utilisant l'utilitaire **wsimport** fourni par le jdk1.6.
- Avant de faire cette opération, nous avons besoin d'apporter une petite modification au niveau du fichier WSDL généré par wsgen. Cette modification concerne l'adresse du web service. Editer BanqueService.wsdl et remplacer la valeur de l'attribut location de l'élément <soap :address> qui se trouve dans l'élément <service>, de **REPLACE_WITH_ACTUAL_URL** par l'adresse réelle du web service : **http://localhost:8686/BanqueService**

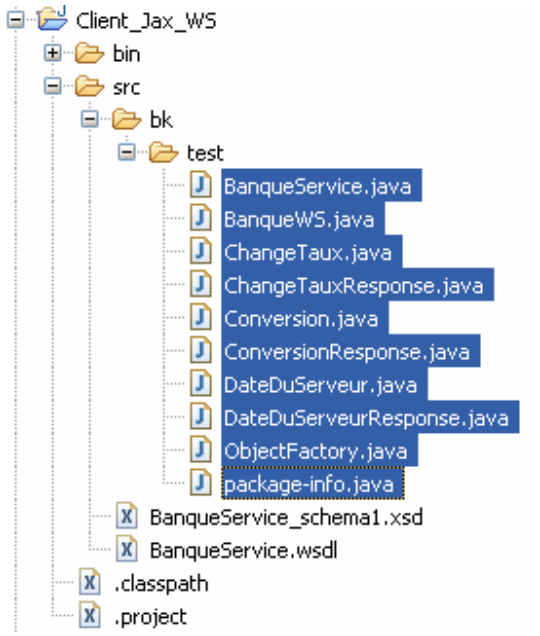
```
<service name="BanqueService">
  <port name="BanqueWSPort" binding="tns:BanqueWSPortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
 </service>
```

```
<service name="BanqueService">
    <port name="BanqueWSPort" binding="tns:BanqueWSPortBinding">
      <soap:address location="http://localhost:8686/BanqueService"/>
    </port>
  </service>
```

- Sur ligne de commande, placez vous dans le dossier src de votre projet. Et taper la commance : **wsimport –s . BanqueService.wsdl**



- Les fichiers suivants seront générés :



- Il est temps maintenant de créer facilement un client java pour notre web service.
- Dans le dossier src créer la classe ClientJWS suivante :

```
import bk.test.BanqueService;
import bk.test.BanqueWS;

public class ClientJWS {

	public static void main(String[] args) {
		BanqueWS service=new BanqueService().getBanqueWSPort();
	System.out.println("Date du serveur:"+service.dateDuServeur());
		double mt=100;
	System.out.println(mt +" Euro ="+service.conversion(mt)+" DH");
	service.changeTaux(11);
	System.out.println(mt +" Euro ="+service.conversion(mt)+" DH");
	}

}
```

- Résultat d'exécution :

```
Date du serveur:2009-01-13T12:53:01.609+01:00
100.0 Euro =1000.0 DH
100.0 Euro =1100.0 DH
```